Recognition of Machine-Printed Arabic Mathematical Formulas

Khazri Kaouther, Afef Kacem, Abdel Belaïd

Recognition of Machine-Printed Arabic Mathematical Formulas

Kaouther Khazri Ayeb and Afef Kacem Echi
University of Tunis, LaTICE-ESSTT
Tunis, Tunisia
kawther.khazri@edunet.tn
afef.kacem@esstt.rnu.tn

Abdel Belaïd
University of Lorraine, LORIA
Nancy, France
Abdel.belaid@loria.fr

***Abstract*—The proposed system recognizes machine-printed Arabic mathematical formulas, extracted from scanned images of clearly printed documents, and outputs the recognition results in MathML format. Two main stages are followed by the proposed system: symbol recognition and symbol-arrangement analysis. A combination of different statistical features (Run length, central Zernike moments, Bi-level co-occurrence, etc.) and K*, an instance-based classifier, have been used to achieve high accuracy for the recognition of mathematical symbols. We defined a set of replacement rules by a coordinate grammar to parse mathematical formulas. We used the coordinate grammar with emphasis on symbol recognizer as well as symbol arrangement analysis. Both ascending and descending parsing scheme, based on operator dominance, has been used to parse the formula. The proposed system provides output in MathML and achieves satisfactory results.**

Keywords—Symbol Recognition; Structural analysis; Notation parser; Mathematical formula interpretation, MathML.

I. INTRODUCTION

Recognizing mathematical formulas is not an easy business and the failure of conventional optical character recognition systems to treat mathematics has several reasons. In fact, mathematics has a number of features which distinguish it from conventional text. These include structure in two dimensions (fractions, super-scripts, sub-scripts, limits, etc.), frequent font changes, different symbols (alphabetic characters, Greek letters, numerals, math operators, etc.) with variable shape (fraction bars, roots, great delimiters, etc.), and substantially differing notational convention from sources to sources. When compounded with more generic problems such as noise and merged or broken characters, they fail to even recognize the symbols adequately partly because they are not on appropriate baseline, are unusual in typeface and size, and do not conform to expectations of text. Notice that the recognition of mathematical formula is more complicated even when all the individual symbols can be recognized correctly. A large part of Arabic books display mathematical formulas using original symbols in a writing running from right to left. A fully automatic system for machine-printed Arabic mathematical formula recognition is proposed here. The paper is organized as follows. In section I, we will quite some characteristics of Arabic mathematics notations and we will show what is special and difficult about recognizing them. The existing mathematics notation recognition literature will be examined in section II. A description of our system can be found in section III. We will close the paper with some experimental results and concluding remarks.

II. CHARACTERISTICS OF ARABIC MATHEMATICAL FORMULAS

Four common styles are used for mathematics within Arabic texts: 1) Moroccan style (see Figure 1), closely related to the French style (see Figure 2), 2) Maghreb style, widely used in North Africa (see Figure 3), 3) Machrek style, generally used in the Middle East (see Figure 4) and 4) Persian style which uses the Arabic script (left to right), but with the mathematical directionality (right to left), similar to the Moroccan style. Figure 5 shows the notation used for limits in Persian style.

$$f(x) = \begin{cases} \sum_{i=1}^{s} x^i & \text{إذا كان} \quad x < 0 \\ \int_{1}^{s} x^i \, dx & \text{إذا كان} \quad x \in E \\ \operatorname{tg} \pi & \text{غير ذلك } (\pi \simeq 3,141 \text{ مع}) \end{cases}$$

Fig. 1. Moroccan style [1].

$$f(x) = \begin{cases} \sum_{i=1}^{s} x^i & \text{si } x < 0 \\ \int_{1}^{s} x^i \, dx & \text{si } x \in \mathbf{E} \\ \operatorname{tg} \pi & \text{sinon (avec } \pi \simeq 3,141) \end{cases}$$

Fig. 2. French style [1].

$$\text{د}(\text{س}) = \begin{cases} \sum_{\text{ن}=1}^{\text{ص}} \text{س}_{\text{ن}} & \text{إذا كان} \quad \text{س} > 0 \\ \prod_{1}^{\text{ص}} \text{س}_{\text{ن}} \text{ ، س} & \text{إذا كان} \quad \text{س} \ni \text{م} \\ \pi \text{ طا} & \text{غير ذلك (مع } \pi \backsimeq 3,141) \end{cases}$$

Fig. 3. Maghreb style [1].

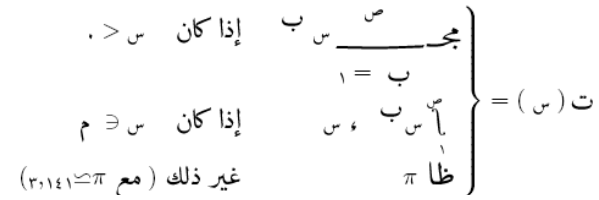

Fig. 4. Machrek style [1].

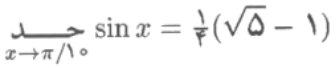

Fig. 5. Persian style [1].

As it can be seen, there are, at least, two main ways for writing Arabic mathematical formulas. In some contexts, mathematical texts use the usual mathematical symbols just as they are in Latin script basic texts. Mathematical expressions flow then from left to right against the stream of the natural language. In other contexts, mathematical texts use specific symbols spreading out from right to left in accordance with the natural language writing. Of course, in both the two mathematical notation systems, the mathematical expressions have exactly the same meaning. Only the way mathematical expressions are presented is different.

Notice that formula interpretation depends on the direction in which it is written. This problem comes from direction inconsistency in handwriting flow. In most of the cases, mathematics is written in the direction opposite to the surrounding text. So, it is important to determine the direction in which mathematical context was entered before proceeding with expression analysis. Notice also that Arabic mathematical formulas require a large number and wide variety of signs (letters are from several alphabets: Arabic, Latin, Greek, etc. in several styles: Naskh, Koufi, etc. for Arabic and Roman, italic, calligraphic, etc. for Latin alphabet) and in different features (lowercase and uppercase, bold). Moreover, Arabic letters vary their form according to their position in the word and some alphabet used in symbolic expressions, appears without diacritical points or signs to mark vowels (see Figure 3). Accents are written in different shapes and sizes. Punctuation exists in several forms (the comma is geared up for a couple of terms or decimal separator and it is oriented low as in French in a decimal (see Figure 3)). Also, there are different forms of numbers: Arab Maghreb numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), Mashrek Arab numbers (٠, ١, ٢, ٣, ٤, ٥, ٦, ٧, ٨, ٩), and Persian numbers (٠,١ , ٢, ٣, ۴, ۵, ۶, ٧, ٨, ٩). Of this, some confusion arise (Zero: ٠ vs. a diacritic point, the Arabic letter ALEF isolated form: ا vs. the Arabic digit one: ١ and the Arabic digit five ٥ vs. the Arabic letter HEH isolated form: ه). In addition, there are two kinds of symbols: literal and mirrored symbols which are used according to the local area. The sum (see Figure 6(a)), product (see Figure 6(b)), the limit (see Figure 6(c)) and factorial (see Figure 6(d)) operators are presented in the two ways.

Letter cursivity of Arabic is also taken advantage of, in a few cases, to define variables using more than one letter. The most widespread example of this kind of usage is the canonical symbol for the radius of a circle نق, which is written using the two letters ن and ق. When variable names are juxtaposed (as when expressing multiplication) they are written non-cursively.

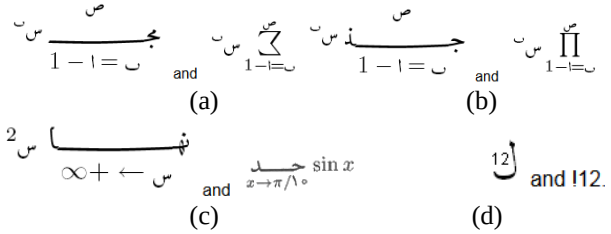

Fig. 6. Literal and mirrored symbols.

Stretched large operators in Arabic notation are usually stretched to the same width as their lower and upper limits (see Figure 7).

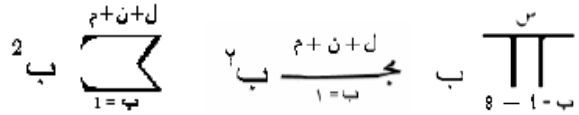

Fig. 7. Stretched large operators.

III. Related Works

Due to the complex characteristic of mathematical Arabic formulas, most of the research concerns online-recognition. In fact, there are few works that delve into offline recognition of Arabic mathematical formulas. Most papers are more concerned with Latin mathematical formula recognition. But they did not put much emphasis on explaining how the mathematical symbols are recognized or how the formula structure is analyzed or the explanations are too tedious and sometimes too ad hoc. In addition, the majority of works are done on some types of formulas with a specific style and typography. They cannot handle all kind of formulas. They generally recognize simple equations but not matrix or system of equations.

In [2], authors briefly described the segmentation and recognition of Simple Arabic Mathematical Equation (SAME) structure and Characters/Symbols in still, gray-scale images. Mainly SAME are numbers, characters and symbols. It is not complex level where there is integration under square root or deferential equation in parentheses at second derivative. It is just main operation operator plus, minus, product and division of variable or numbers with variable in any order. At the segmentation step, the system applies threshold for gap detection between SAME parts. Secondly it counts the end points for each connected component in that part for characters and symbols position determining, the expression structure decoding and define the operator location. At last, the system applies Self–organized map recognizer on the extracted feature from connected components of SAME parts. The system is tested on a set of handwritten and printed expressions and achieves promising results.

As discussed by [3], quite a number of mathematical expression recognition systems obtained the structure without parsing. Instead, some procedurally-coded rules were used while others applied parsing techniques with a range of

variations. However, it should be noted that most methods for the structural analysis of mathematical expressions are actually based on some kinds of syntax defined implicitly or explicitly. For simple expressions, both ways should do the job well. The situation changes if we try to recognize more complex expressions. Rather than adding ad hoc procedurally-coded rules to the system and yet still being uncertain of the correctness of the structural analysis module, explicit rules in a parser may provide a clearer and more concise form for formal verification. The problem of creating a system which is both efficient and sufficiently flexible to recognize complex relationships remains an open problem. Flexible refers to the ability to handle a large range of symbol placements. In this paper, we propose a system based on a syntactic parser. It starts the parsing by looking for the most important operator in the formula and attempts to partition it into sub-formulas which are similarly analyzed. The formula parser works in conjunction with a symbol recognizer as it will be explained in the following section.

IV. PROPOSED SYSTEM

The proposed system consists of two main stages: symbol recognition and symbol-arrangement analysis. The former converts the input image into a set of symbols. The latter analyzes the spatial arrangement of this set of symbols to recover the information content of the given mathematical formula. We are interested by formulas written according the Mashrek style (see Figure 4). An isolated symbol recognition scheme consisting of a K* classifier based on statistical features is used [4]. For the recognition of meaningful arrangement of the recognized symbols, a structural analysis, based on the bounding box coordinates, is used. A set of coordinate grammar rules is served to parse formulas and convert them into their corresponding MathML code strings. We will explain how our system proceeds to achieve the above steps.

A. Symbol Recognition

To describe mathematical symbols, we extracted the following features:

- Run-length histogram features which take into account the number of successive black pixels in one or more directions (horizontal, vertical, major diagonal and minor diagonal). These features can effectively discriminate the directions, areas and geometrical shapes of the symbols.
- Zernike moments which have rotational invariance and are accurate descriptors even with relatively few data points. They are often used to capture global features of an image. They have proven their superiority over other functions moments in their description ability and robustness to noise and distortion.
- Bi-level co-occurrence: A co-occurrence count is the number of times a given pair of pixels occurs at a fixed distance and orientation.
- Another feature is considered which computes the white pixel's portion in the symbol image.

For symbol classification, we used an instance-based classifier, named K*, where the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. Note that K* differs from other instance-based learners in that it uses an entropy-based distance function. For more information on K*, see [4].

We used a database composed of 47 symbol classes (some variable and function names, arithmetic operators, literal and mirrored symbols, Mashrek Arab numbers, etc.) with 60 samples par class. The symbol recognizer achieves a rate of 98.48%. Although the symbol recognizer achieved a good accuracy, its failure to distinguish certain common symbols would be bothersome to any serious use. In fact, certain distinct symbols are in close resemblance such Arabic zero and diacritic point (٠, .), minus sign and horizontal fraction bar (-, —), Arabic digit one and Arabic letter Alif (١, ا), etc. Observing the event of confusion, we remark that confused symbols have roughly similar morphologies. We consider some of the misrecognitions to be too difficult for any classifier to resolve without considering symbol context.

B. Structural analysis

This subsection explains how to parse mathematic structures based on lexical, geometrical and syntactical analysis.

1) Lexical analysis: Table 1 gives the input characters, strings and symbols to recognize common functions, limit, new function, square root, integral, sum and product, arithmetic expressions, fractions and their corresponding syntactic category. The terminal alphabet of the syntax consists of the syntactic categories, listed in the first column of this table. For multi-parts symbols or words, some processing is needed to group symbols, letters or function names properly into units. For example, some pieces multi-part symbols are joined together by vertical grouping to form symbols or letters or function names like '=', '≤', '≥', أ, ب, ت, ث, خ, ح, ج, ذ, ز, ض, ش, غ, ظ, ق, نها, ن, مجد, مج, etc. We considered a letter followed by an open delimiter and closest to its left neighbor as a new function's name. A digit or sequences of digits, which are horizontally adjacent, should compose unsigned integers. Unsigned floats consist of unsigned integers separated by a decimal point.

2) Geometrical analysis: It is worth noting that spatial relationships are especially critical for the pre-processing step during the lexical analysis and the recognition of implicit operators such as subscripts, superscripts and implied multiplication. Geometric criteria are here used to check if a set of components has possible links between them. Ten relations: Left (L), Right (R), Above (A), B (Below), LS (Left Superscript), RS (Right Superscript), Ls (Left subscript), Rs (Right subscript), I (Inside) and D (Delimited) are defined to describe spatial structure in mathematics notation. These spatial relations are also used at this level to remove confusions between morphological similar symbols (diacritic

point and Arabic digit zero, minus sign and horizontal fraction bar, etc.). For differentiating between them, a context have to be defined. For example, in order for a symbol to be considered as a fraction bar, it should have no empty parts above and below it. To separate between diacritic point and Arabic zero, we can see if the regions above and below contains letters or function names. Where letter or function name found, it is seen as a diacritic point.

TABLE I. SYMBOL LABELING

Labels	Symbols sets	Designation
SS	∑, مجـ, مجـــ	Literal and mirrored Sum symbols
PS	∏ , جذ, جـــذ	Literal and mirrored product symbols
RS	√	Square Root
IS	∫	Integral
FB	—	Horizontal Fraction Bar
DL	(,), {, }, [,], |	Delimiters
OP	=, <, >, ≥, ≤, +, ×, /	Operators
FL	←	Flèche
Unsigned-integer	٣ ,١٢ , etc.	Unsigned Integers
Unsigned-float	٣.٢, etc.	Unsigned Floats
Letter	أ, ب,ت, etc.	Letters
FN	جا, جتا, ظا, ظتا, حتا, حا طا, طتا	Function Names.
NF	ق(س), etc.	New Function
LM	نها	Limit

3) Syntactic analysis: Our system begins the parsing by looking for the start operator and attempts to partition the formula into sub-formulas which will be similarly analyzed. The location of the operator from which interpretation begins in unambiguous expressions is considered as hard convention of mathematics notation. Expressions are generally interpreted beginning with their rightmost operator but some exception include fraction bars not being rightmost in their associated sub-expression or limit symbols with overlapping limits. In these cases, operator dominance as defined by [5] may be used to locate the dominant operator in the rightmost sub-expression, from where the interpretation begins. But, there is a number of cases where it is impossible to determine the start operator of an expression because either it is impossible to determine operator dominance or the range of operators is unclear (such as when a fraction bar overlaps a symbol lightly, making it unclear whether that symbol is an argument of the division or an adjacent term). In the following, we introduce the proposed coordinate grammar. We then explain how to choose the start operator. We finally give an illustrative example to explain how to analyze formula structure.

*a) Coordinate Grammar:*Each production rule maps a set of symbols, located at given coordinates, into a new set of symbols whose coordinates are computed by a set of functions associated with the given production. The proposed coordinate grammar is used here with emphasis on symbol recognizer as well as symbol arrangement analysis. As shown in table 2, with each rule, our system associates contexts, to be checked each time the rule is chosen in the input process and actions to be performed. These actions convert mathematical formulas into their corresponding MathML codes. To have easy communication between the actions and the parser, the compounds of the right side of a rule are numerated from 1 to n reading from the right to the left. Note that the syntax is restricted to commonly used arithmetic notations. We cannot use the full generality of mathematics notation, even if it can in principle be typeset, because our notion of mathematics grammar is necessarily limited to those constructions we are aware of.

b) Start Operator choice: It has particular importance. It is not necessary the beginning or the end of the formula. It can be an explicit, represented by a symbol, or implicit operator like subscripts, superscripts or implicit multiplication. We have to use the concept of operator dominance and precedence to choose the start operator especially when the formula contains many operators which are not lined up. Notice that, the operator O_1 dominates the operator O_2 if O_2 lies in the range of O_1. The range of an operator involves all possible emplacements of its operands. The process of finding the start operator of an expression takes the set of symbols as input and it returns its position in case of an explicit operator. If the start operator is implicit, it returns position of the column that split the expression in two sub-expressions. Below are the followed steps:

1. Compute, for each operator *O*, how many times it has been dominated by the other operators. Let call *dom(O)* this number,
2. The start operator is the one that is less dominated. In case of expression in Figure 8, there are two symbols: equal sign and square root but the start operator corresponds to the equal sign since (dom(OP)=0 whereas dom(RS)=1),

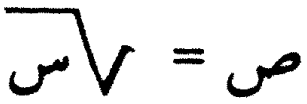

Fig. 8. Operators with different dominance.

3. If many operators have the same dominance, we choose the prior one using operator precedence rules. We start, if they exist, by operators of comparison (=, ≠,≤, ≥, etc.), then parentheses which have higher precedence than the other operators such as superscript, subscript, unary operator (-, ±, etc.) and arithmetic operators (*, /, +, -, etc.). In Figure 9, '+' and '=' have the same dominance but as '=' is prior than '+' and '*', we choose it as start operator.

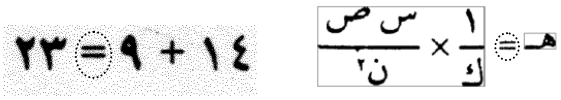

Fig. 9. Operators with same dominance and different precedence.

4. If many operators have the same dominance and the same precedence, we choose the most right one. In expression of Figure 10, the first and the latter equal signs have the same dominance and precedence.

Fig. 10. Operators with the same dominance and precedence.

5. The area occupied by the start operator, with its range, should correspond to the entire formula otherwise it is about an implicit multiplication. In such case, we return the column that splits the formula into two sub-formulas using the leftmost operator (see Figure 11).

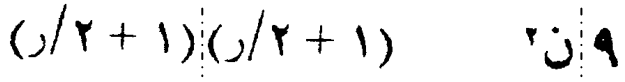

Fig. 11. Implicit multiplication of two sub-expressions.

6. If no operator is found, as shown in the example of Figure 12, then it is about an implicit multiplication of variables. We then return the column that spilt the formula into two sets of letters.

Fig. 12. Implicit multiplication of variables.

V. EXPERIMENTS

We carried experiments on a database of 110 machine-printed Arabic mathematical formulas. We compared between our system output and the ground truth. Figure 13(b) shows MathML code, returned by our system whereas Figure13(c) displays what the system should return for the input formula presented in Figure 13(a). As it can be seen, there is a mistake (colored in red in Figure 13(b)) in coding the Arabic letter 'س'. In fact, this letter has been confused with the Arabic letter 'ص'

(a)

```
<html><head></head><body><
math
xmlns="http://www.w3.org/19
98/Math/MathML"
mode="display"
dir="rtl"><mrow><mi>&#x0642
;</mi><!--
<mo>&ApplyFunction;</mo>--
><mfenced open="("
close=")"><mi>&#x0635;</mi><
/mfenced><mo>=</mo><mn>&
#x0660;</mn></mrow></math
></body></html>
```

(b)

```
<html><head></head><body><
math
xmlns="http://www.w3.org/19
98/Math/MathML"
mode="display"
dir="rtl"><mrow><mi>&#x0642
;</mi><!--
<mo>&ApplyFunction;</mo>--
><mfenced open="("
close=")"><mi>&#x0633;</mi><
/mfenced><mo>=</mo><mn>&
#x0660;</mn></mrow></math
></body></html>
```

(c)

Fig. 13. MathML code returned by our system.

The average rate of mathematical interpretation of our system is computed by dividing the total of interpretation rates of all formulas by their number. The interpretation rate of each formula is calculated as follows:

Formula's Interpretation Rate=1-Error Rate

To determine the Error Rate, we compare between the output of our system with a ground truth using the Levenshtein distance. This distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other. We used this metric to consider and pinalize all possible errors in formula MathMl code output: 1) insertion of a single symbol (If truth code = *uv*, then inserting the symbol *x* produces *uxv*), 2) deletion which changes the truth code *uxv* to *uv*) and 3) substitution of a single symbol *x* for a symbol $y \neq x$ which changes the truth code *uxv* to *uyv*. For each interpreted formula, the number of errrors (substitutions, deletions and additions) is divided by the length of the truth code string to get the Error Rate. For the example in Fig.13, the error rate is equal to 1/263, so its interpretation rate is 1-(1/263) which is near to 1. Tested on handered formulas, we achieved a very satisfacory average rate of formula interpretation which is above 95%.

VII. CONCLUSION

In this paper, we address the problem of recognizing Arabic mathematical formula. For symbol recognition, we used a successful recognition method based on *k** with a combination of different statistical features. Our system provides a success rate above 95%. To improve this work, we plan to elaborate tests of efficiency and performance of the proposal system on a larger database of symbols, since the development of efficient methods depends heavily on the database availability of large size permitting to test the performance, robustness, reliability of the proposed system and conduct meaningful statistical tests to compare against each other. We also explained how our system interprets mathematical formulas using a coordinate grammar which provides a clear and well-structured approach. Then the structure representation is encoded in MathML which facilitates automatic processing, searching and indexing, and reuse of mathematical documents. The overall system has shown its efficiency on a reasonable number of practical mathematical formulas. Further work is required to extend this method to treat noisy, more complex formulas and even handwritten formulas to confirm the efficiency and the robustness of our system.

REFERENCES

[1] Benjamin P. B. and Fateman J., "Optical character recognition for typeset mathematics", in proceeding of International Symposium on symbolic and Algebraic computation, pp. 348-353, 1994.

[2] Khalifa M. and Bing Ru Y., "A Hybrid Segmentation System of Offline Arabic Mathematical Expression Recognition", Canadian Journal on Image Processing and Computer Vision, Vol. 2, No. 4, pp. 30-35, 2011.

[3] Blostein D. and Grbavec A., *Recognition of Mathematical Notation*, Handbook of character recognition and document image analysis, Eds. H. Bunke and P.S.P. Wang, world scientific publishing company, pp. 557-582, 1997.

[4] John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 108-114, 1995.

[5] Chang S. K. Chang, "A method for the structural analysis of two-dimensional mathematical expressions", in *information sciences*, vol. 2, pp.253-272, 1970.

TABLE II. PRODUCTION RULES AND MATHML ENCODING

Production rules	Assumption	MathML Encoding
R_1: $E^0 \rightarrow E^2$ ***OP*** E^1	E^1=R(***OP***)≠ϕ , E^2=L(***OP***)≠ ϕ OP.code='+'|'*'|'/'|'>'|'<'|'≤'|'≥', '='	E^0.code= <***mrow> E^2.code <mo>OP.code</mo> E^1.code </mrow*** >
R_2: $E^0 \rightarrow E^2 E^1$	E^1=R(E^2)	E^0.code=<***mrow>E2.code<mo>&invisibletimes;</mo> E^1.code</mrow***>
R_3: $E^0 \rightarrow E^2$ ***FL*** E^1	E^1=R(***OP***)≠ϕ , E^2=L(***OP***)≠ ϕ FL.code='←'	E^0.code=<***mrow>E2.code<mo>→</mo> E^1.code</mrow***>
R_4: E → T ***OP***	T=L(***OP***)≠ ϕ , R(***OP***)= ϕ ***OP***.code='+'|'–'|'±' A(***OP***)=B(***OP***)=ϕ	E.code=<***mrow> T.code <mo>OP.code</mo></mrow***>
R_5: E → T		E.code=T.code

R_6: $E^0 \rightarrow V\ Letter\ E^{1^N}_S\ IS$	S=ϕ|B(***IS***)|Rs(***IS***)|Ls(***IS***)
	N=ϕ|A(***IS***)|RS(***IS***)|LS(***IS)***
	L(***SI***)=E^1≠ϕ , R(***Letter)***=E^1≠ϕ
	L(***Letter)***=V≠ϕ
	Letter.code=' ،'|'ↄ'
		if (N≠ϕ et S≠ϕ) then E^0.code = ***V.code<mo>ↄ</mo> E^1.code <munsubsup> S.code N.code<mo>∫</mo> </munsubsup***>
		else if (S=N=ϕ) then E^0.code = ***V.code<mo>ↄ</mo> E^1.code <mo>∫</mo***>
		end if

R_7: $E^0 \rightarrow E^{1^N}_S\ SS$	N=ϕ | A(***SS***) | RS(***SS***) | LS(***SS***) S=B(***SS***)|Rs(***SS***)|Ls(***SS***) I(***SS***)= ϕ , E^1=L(***SS***)≠ ϕ	if (N≠ϕ) then E^0.code= ***E^1.code <munderover>S.code N.code <mo>&asum ;</mo> </munderover*** > else E^0.code= ***E^1.code <munder>S.code<mo>&asum ;</mo> </munder*** > end if
R_8: $E^0 \rightarrow E^{1^N}_S\ SP$	N=ϕ | A(***SP***) | RS(***SP***) | LS(***SP***) S=B(***SP***)|Rs(***SP***)|Ls(***SP***) I(***SP***)= ϕ , E^1=L(***SP***)≠ ϕ	if (N≠ϕ) then E^0.code= ***E^1.code <munderover>S.code N.code <mo>&asum ;</mo> </munderover*** > else E^0.code= ***E^1.code <munder>S.code<mo>&asum ;</mo> </munder*** > end if
R_9: $E^0 \rightarrow$ E^1 ***FB*** E^2	E^1=A(***FB***)≠ ϕ , E^2=B(***FB***)≠ ϕ	E^0.code = <***mfrac>E^1.code E^2.code </mfrac*** >
R_{10}: $E_0 \rightarrow E^1\ RS^R$	R=ϕ|RS(***RS***) E^1=I(***RS)***≠ ϕ	if (R≠ϕ) then E^0.code= ***<mroot> R.code E^1.code </mroot> else E^0.code=<msqrt> E^1.code </msqrt***>end if
R_{11}: $E^0 \rightarrow$ ***DL^2*** E^1 ***DL^1***	E^1=D(***DL^2***, ***DL^1***)	E^0.code=***<mrow><mfenced open="DL^2.code" close="DL^1.code">*** E^1.code ***</mfenced></mrow*** >
R_{12}: $E^0 \rightarrow^X E^1$	X=LS(E^1)| ϕ	if **(**X=ϕ) then E^0.code=***<msup><mrow> E^1.code </mrow><mrow> X.code</mrow> </msup> else E^1.code*** end if
R_{13}: $E^0 \rightarrow E^1\ ^X$***FN***	X=ϕ|LS(***FN***) E^1=L(***FN***)≠ ϕ	if (X≠ϕ) then E^0.code= ***E^1.code <msup><mrow>FN.code</mrow><mrow> X.code</mrow> </msup***> else E^0.code = ***E^1.code NF.code*** end if
R_{14}: $E^0 \rightarrow E^1$ ***NF***	E^1=L(***NF***)≠ ϕ	E^0.code= ***NF.code<&Applyfunction ;> E^1.code***
R_{15}: $E^0 \rightarrow E^1$ ***LM*** S	S=B(***LM***) | ϕ E^1=L(***LM***)≠ ϕ	if (s≠ϕ) then E^0.code= ***E^1.code <mo>&alim</mo><munder> S.code </munder*** > else E^0.code= ***E^1.code <mo>&alim</mo*** > end if
R_{16}:T → V		T.code = ***V.code***
R_{17}:T →***unsigned-float***		T.code = <***mn> unsigned-float.code</mn***>
R_{18}:T→***unsigned-integer***		T.code = <***mn> unsigned-integer.code</mn***>
R_{19}: N → E		N.code = ***E.code***
R_{20}: N → ***Letter OP***	***OP***.code='+'|'–' ***Letter***.code='∞' R(***Letter***)= ϕ|***OP***	if D(lettre) ≠ ϕ then N.code = ***<mrow><mo>∞</mo>*** <***mo>OP.code</mo> </mrow*** > else N.code = <***mo>∞</mo***>endif
R_{21}: N → ε		
R_{22}: S → E ***OP*** V	***OP.***code='='|'<'|'≤'|'>'|'≥'|'∈'|'←' V =R(***OP***)≠ϕ, E=L(***OP***)≠ϕ	S.code=<***mrow>E.code<mo>OP.code</mo>*** ***V.code</mrow***>
R_{23}: S → V		S.code =***V.code***
R_{24}: R → V		R.code=***V.code***
R_{25}: R→***unsigned-integer***		R.code=<***mn> unsigned-integer.code</mn***>
R_{26}: R →ε		
R_{27}: X →E		X.code=***E.code***
R_{28}: X →ε		
R_{29}: D^0 →D^1***Letter*** E	***Letter.code=','*** D^1=R(***Letter***)≠ϕ,***E***=L(***Letter***)≠ϕ	D.code= ***<mrow>D.code <mo>OP.code,</mo> E.code</mrow*** >
R_{30}: D → E		D.code=***E.code***
R_{31}: V → D ***Letter***	D=Ls(***Letter***)≠ϕ	V.code=<***msub> D.code <mi> Letter.code</mi></msub***>
R_{32}: V →***Letter***		V.code=<***mi> Letter.code</mi***>